The Grid File

An Adaptable, Symmetric Multikey File Structure

Vortrag von Markus Apell im Rahmen des Seminars
„Support for Non-Standard Datatypes in DBMS“

Inhalt:

1. Einführung

Eine symmetrische, sich anpassende Multikey-Datenstruktur

In Datenstrukturen wie Baumstrukturen, sortierte Listen oder Hashfiles werden die Datensätze anhand von Singlekeys organisiert, also anhand eines Attributes, dem sogenannten Primärschlüssel. Damit werden Operationen wie FIND, INSERT und DELETE ermöglicht. Bei der Entwicklung früherer Strukturen wurden Einfüge- und Löschoperationen oft nicht für wichtig erachtet und vernachlässigt. Mit sogenannten Überlaufbereichen wurde dem Problem auf Kosten der Leistung begegnet. Für manche Strukturen konnten noch elegante Lösungen gefunden werden, wie beispielsweise die balancierten Bäume, die ihre Form anpassen.
Symmetrisch ist eine Struktur, wenn jedes Attribut wie ein Primärschlüssel behandelt wird. Ein anschauliches Beispiel ist das Telefonbuch: Will man wissen, welche Telefonnummer ein Herr Nievergelt hat, so schaut man unter dem Buchstaben „N“. Wenn man aber wissen will welcher Name sich hinter der Telefonnummer 09876/54321 verbirgt, wird es einige Zeit brauchen, bis man die Antwort erhält, weil man sich im schlimmsten Fall jeden Eintrag ansehen muss. Offensichtlich ist nur nach einem Attribut geordnet (Asymmetrie). Im symmetrischen Fall sind die Kosten für eine Operation nicht vom Attribut abhängig. Der Zeitaufwand der beiden Anfragen wäre im symmetrischen Fall nicht derart unterschiedlich.
Anpassbarkeit: Eine Datenstruktur passt ihre Form dem Inhalt an, dadurch wird erreicht, dass die durchschnittliche Speicherplatzauslastung und die Zugriffszeiten im ganzen File nahezu gleich bleiben. Die oben genannten balancierten Bäume sind dafür ein gutes Beispiel.
Multikey Access: Zugriff anhand eines oder der Kombination von mehreren Attributwerten. Viele Multikey-Strukturen sind nur Weiterentwicklungen von Singlekey-Strukturen. Selbstsprechend sind diese weder elegant noch ressourcenschonend.

Die zwei Prinzipien

Neben der Speicherauslastung sind die Zugriffszeiten ein wichtiges Qualitätsmerkmal. Die beiden wichtigsten Faktoren der Zugriffszeit sind die Anzahl und die Geschwindigkeit der Plattenzugriffe. Bereits sehr früh versuchte man, die Anzahl der Plattenzugriffe einzuschränken, und so entstanden in den 50ern die folgenden beiden Prinzipien:
'Two-Disk-Access' Prinzip: Bei einer voll spezifizierten Anfrage muss ein Datensatz nach mindestens zwei Plattenzugriffen zurückgegeben werden.
'Efficient Range Queries with Respect to all Attributes': Datensätze, die sich in den Werten ihrer Attribute ähneln, sollen auch im physischen Speicher so nahe wie möglich beieinander liegen. Auf diese Weise wird sichergestellt, dass die Ergebnisse von Bereichsabfragen schneller gefunden werden. Eine Bereichsabfrage könnte folgenderweise lauten: „Zeige mir alle Personen die zwischen 20 und 25 Jahre alt sind“.
Beide Prinzipien wurden selten erfolgreich umgesetzt. Bei den Singlekey-Strukturen beispielsweise nicht wegen der Überlaufbereiche oder bei den bisherigen Multikey-Strukturen nicht, da für jedes weitere Attribut ein erneuter Zugriff fällig wurde.

Wie das Grid File die drei Eigenschaften Symmetrie, Anpassbarkeit und Multikey Access realisiert und auch die beiden Prinzipien erfüllt, werde ich im Folgenden erläutern.

2. Konzept und Komponenten des Grid Files

Das Prinzip

Ein Datensatz ist ein Punkt in einem mehrdimensionalen Raum. Dieser Datenraum wird dynamisch durch ein orthogonales Gitter aufgeteilt. Durch diese Aufteilung entstehen die sogenannten Grid Blocks. Jede Dimension des Datenraums entspricht einem der Attribute der abzuspeichernden Daten. Der Wertebereich der Attribute ist auf seiner Dimension linear angeordnet.

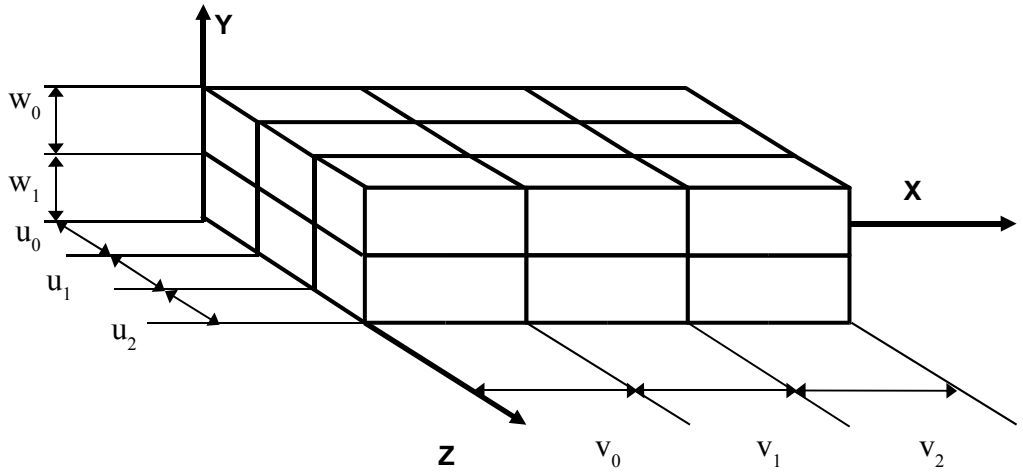

Abbildung 1: Beispiel für einen dreidimensionalen Datenraum mit den Attributen X, Y und Z.

Für einen dreidimensionaler Datenraum mit den Attributen *X*, *Y* und *Z* erhält man eine Zellpartition ('grid partition') $P = U \times V \times W$, die jeweils in die Abschnitte $U = (u_1, u_2 \ldots u_l)$, $V = (v_1, v_2 \ldots v_m)$, $W = (w_1, w_2 \ldots w_n)$ eingeteilt sind. Daraus ergibt sich ein dreidimensionaler Datenraum in Form eines Würfels mit $l \times m \times n$ Grid Blocks.

Die Realisierung

Die Datensätze der Grid Blocks werden in den Buckets gespeichert. Diese Speichereinheit liegt auf der Platte und hat eine festgelegte Kapazität, die in der Praxis zwischen 10 und 1000 Datensätzen liegt. Bei dieser Menge an Daten ist die interne Ordnung von geringerer Bedeutung und daher in der Regel linear. In einem Bucket liegen mindestens die Datensätze eines Grid Blocks. Es können aber bis zur vollen Ausnutzung der Kapazität auch die Datensätze weiterer Grid Blocks gespeichert werden. Dennoch gilt wegen des 'Two-Disk-Access' Prinzip zu beachten, dass alle Datensätze eines Grid Blocks im selben Bucket gelagert sind. Damit diese dynamische Korrespondenz zwischen den Grid Blocks und den Buckets aufrecht erhalten wird, bedarf es eines Managementsystems, dem sogenannten Grid Directory. Das Grid Directory ist das Herz der gesamten Grid File Struktur. Ursprünglich war vorgesehen, dass dieses komplett im Arbeitsspeicher liegt. Damit bei größeren Datenmengen nicht zu viel Arbeitsspeicher verloren geht, wurde das 'Resident Grid Directory' entwickelt, bei dem nur eine abgespeckte Version des Directory im Speicher liegt und der für eine Anfrage relevante detaillierte Teil nachgeladen wird.

Das Grid Directory

Das Grid Directory besteht aus zwei Teilen. Zum einen gibt es das Grid Array, das den Datenraum repräsentiert. Es hat *k* Dimensionen, wobei *k* die Anzahl der Attribute der Datensätze ist. In diesem Array werden die Zeiger abgespeichert, die auf das Bucket zeigen, in welchem die gesuchten Datensätze liegen. Zum anderen gibt es *k* Skalierungsvektoren ('linear scales'). Sie entsprechen den Koordinatenachsen des Datenraums, denn sie teilen die einzelnen Dimensionen ein.

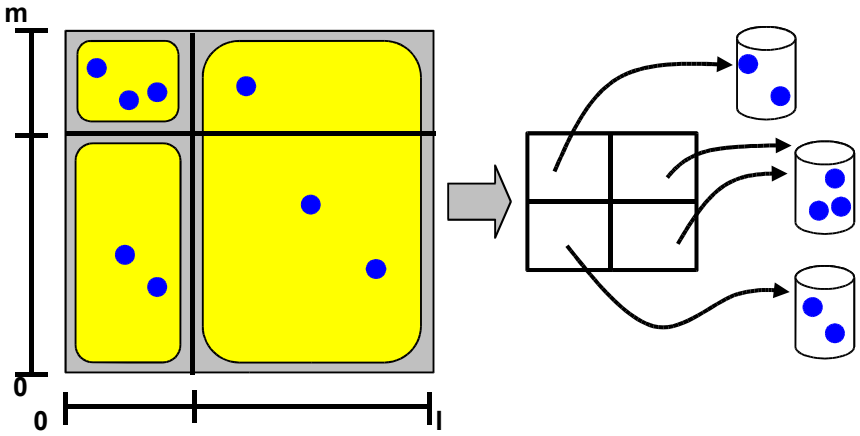

Abbildung 2: Ein zweidimensionaler Datenraum mit den Skalierungsvektoren und zu Regionen (gelb) verbundenen Grid Blocks und den dazugehörigen Zeiger auf die Buckets.

Teilen sich zwei oder mehrere Grid Blocks ein Bucket so nennt man diese eine Region. Hierbei lässt man nur dann eine Verschmelzung eines weiteren Grid Blocks zu, wenn die resultierende Region auch wieder ein Rechteck (für $k = 2$) ergibt. Diese Eigenschaft wird Konvexität genannt und muss stets erfüllt sein. In *Abbildung* 2 sind die beiden rechten Grid Blocks zu einer konvexen Region verschmolzen.
So erlaubt nun das Grid Directory mit einem Grid Array *G* und Skalierungsvektoren *X* (0...l), *Y* (0...m) Operationen wie einen unmittelbaren Zugriff *G* (*x*, *y*) oder die Abfrage nach dem benachbarten Element, wie zum Beispiel *nextabove x* ← *(x + 1)*. Weiter lässt sich die Regel ableiten
$G(x', y') = G(x'', y'')$ mit $x' \leq x \leq x''$ und $y' \leq y \leq y''$, dann $G(x', y') = G(x, y) = G(x'', y'')$.

Der Datensatzzugriff

Betrachten wir nun folgende Anfrage:
In einem zweidimensionalen Datenraum sind Daten mit den Attributen *Jahr* und *Initialen* mit den Wertebereichen [0, ... , 2004] und [*a*, ... , *z*] abgelegt. Als Skalierungsvektoren ergeben sich *X*=(0, 1000, 1500, 1750, 1875, 2004) und *Y*=(*a*, *f*, *k*, *p*, *z*). Nun ist der Datensatz mit dem Jahr *1980* und der Initiale *w* gesucht (vgl. *Abbildung* 3). Oder als SQL-Anfrage:

```
SELECT *
    FROM G
    WHERE Jahr = „1980“
        AND Initiale = „w“
```

Als erstes wird anhand der Skalierungsvektoren bestimmt in welchem Grid Block der Zeiger zum richtigen Bucket liegt. In unserem Beispiel ist es jeweils die letzte Einteilung. Dieses

Bucket wird geladen und der Inhalt durchlaufen bis der Eintrag gefunden ist. Wie bereits erwähnt können in diesem Bucket auch die Datensätze anderer Grid Blocks liegen. Offensichtlich wird bei dieser Anfrage das 'Two-Disk-Access' Prinzip eingehalten.
Bei den Bereichsabfragen ist der Ablauf ähnlich. Will man nun einen ganzen Satz von Daten erfragen, die innerhalb eines Bereiches liegen, so betrachtet man wieder die Skalierungsvektoren. Hierbei macht man sich auch wieder die lineare Ordnung der Dimensionen zunutze. Liegt nun eine Region komplett in dem abgefragten Bereich, so kann man alle dazugehörigen Datensätze ausgeben. Die Datensatzmenge der Regionen, die von dem Bereich geschnitten werden, müssen dagegen erst überprüft werden.

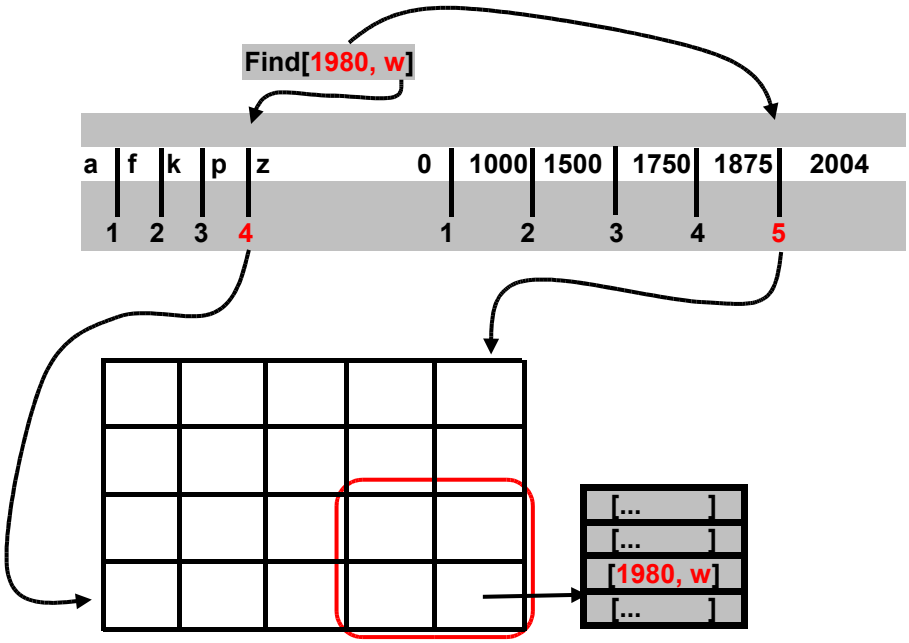

Abbildung 3: Eine exakte Anfrage nach einem einzelnen Datensatz benötigt zwei Plattenzugriffe.

Mit dem Einfügen oder Löschen von Daten verhält es sich ähnlich. Über das Grid Directory wird das entsprechende Bucket gefunden und geladen. Der neue Datensatz wird in das Bucket geschrieben, ein zu löschender Datensatz wird aus dem Bucket entfernt.

3. Dynamik des Grid File

Wie oben erwähnt ist Kapazität eines Buckets festgelegt. Folglich würde nach einigen Einfüge- oder Löschoperationen das Bucket überlaufen oder derart leer sein, dass eine sehr schlechte Speicherplatzauslastung auftreten würde.

Teilungsoperationen

Nehmen wir nun den Fall, das eine Bucketbelegung zu groß wird. Der Inhalt des einen Buckets muss nun auf zwei verteilt werden, dabei wird dessen Region und damit der gesamte Datenraum geteilt. Oft kann es auch vorkommen, dass dabei die Skalierungsvektoren des Grid Directory neu eingeteilt werden müssen (Siehe auch *Abbildung* 4).
Offen bleibt die Frage welche Dimension des Datenraums geteilt werden soll. Eine Möglichkeit wäre, die Dimensionen in zyklischer Reihenfolge zu wählen. Eine andere Möglichkeit ist die Wahl der Dimension mit der gröbsten Einteilung. Generell gilt: Eine bereits bestehende Teilung ist vorzuziehen.

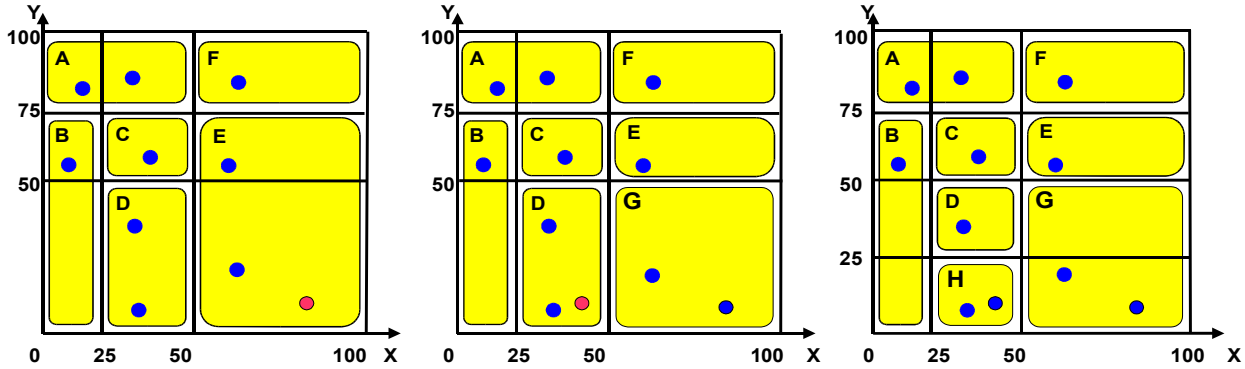

Abbildung 4: Beispiel für Teilungsvorgänge. Die Bucketkapazität ist 2 und es werden die Datensätze (90, 10) und (40, 15) eingefügt.

In *Abbildung* 4 ist ein solcher Vorgang anschaulich dargestellt. Nachdem der Datensatz (90, 10) eingefügt wurde, läuft das Bucket der Region E über, da die Bucketkapazität auf 2 beschränkt sein soll. Ein Teil seines Inhaltes wird in das neue Bucket G kopiert und da es bereits eine geeignete Einteilung im Grid Array gibt, wird diese auch übernommen. Beim Einfügen des zweiten Datensatzes (40, 15) ist eine Neueinteilung eines Skalierungsvektors unvermeidlich. In diesem Fall läuft das Bucket der Region D über und ein neues Bucket H entsteht. In unserem Fall wurde die Dimension Y neu eingeteilt.

Verschmelzungsoperationen

Betrachten wir den Fall, wenn ein Bucket zu leer wird. Dieses Bucket sollte nun mit einem anderen relativ leeren Bucket zusammengefasst werden. Dabei kann man schon vorweg nehmen, dass auf eine Verschmelzung von Sektionen des Grid Directory verzichtet werden kann, da ein baldiges Wachstum und die damit verbundene Teilung von Regionen wahrscheinlich ist.
Welches Bucket ist geeignet, damit die entstehende Region auch wieder konvex ist? Zwei Systeme versuchen, das zu bestimmen: Das 'Buddysystem' erlaubt einer Region nur Verschmelzungen mit Regionen, von denen es abgeteilt wurde. Das 'Nachbarschaftssystem' gestattet Verschmelzungen mit allen Regionen, die benachbart sind, und wenn die Ergebnisregion wieder konvex ist. Während das Buddysystem deutlich weniger Verschmel-

zungsmöglichkeiten zulässt (vgl. *Abbildung* 5), ist das Nachbarschaftssystem anfälliger für einen 'Deadlock', also für die Situation, in der es für eine Region keine benachbarte Region mehr gibt, mit der durch Verschmelzung eine konvexe Region entstehen könnte.
Als nächstes muss wie bei den Teilungsoperationen festgestellt werden, welcher Kandidat Priorität hat, also welche Dimension vorzuziehen ist.
Eine weitere wichtige Frage ist die nach dem Verschmelzungsgrenzwert, also die Belegung des resultierenden Buckets. Die Simulation ergab, dass eine Belegung von 70% optimal ist.

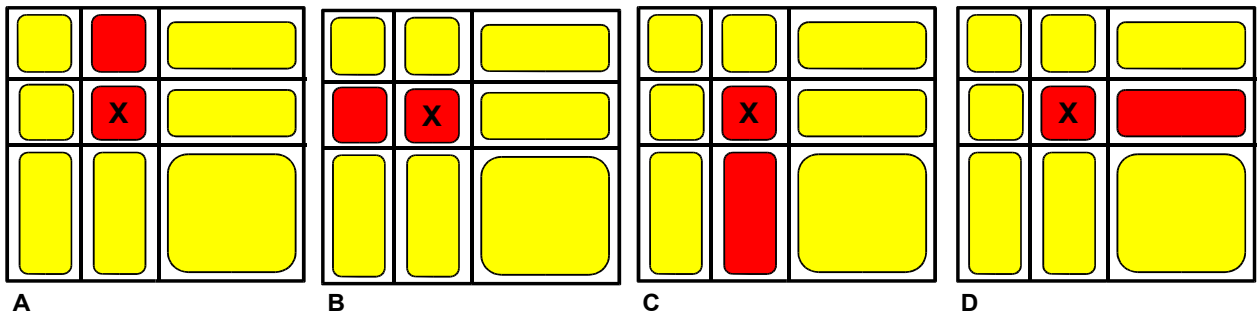

Abbildung 5: Mögliche Regionen, mit denen Region X verschmolzen werden kann. Nach dem Buddysystem käme nur Beispiel A und B in Frage. Beim Nachbarschaftssystem wären alle vier Beispiele denkbar.

4. Die Simulation

Um festzustellen, ob das Grid File tatsächlich die gewollte Performanz bietet, kann man nicht auf die bekannten Analysemodelle der Singlekey Strukturen zurückgreifen, da diese nicht für eine Mehrdimensionalität geeignet sind. Einige Parameter, wie zum Beispiel die diversen Verschmelzungs- und Teilungsmethoden, machen das Grid File zu kompliziert für ein mathematisches Modell. So bleibt die Beobachtung einer Simulation, bei der das Augenmerk auf der Abschätzung der Bucketbelegung und Directorygröße, der Visualisierung der Geometrie der Bucketregionen und der Bewertung der Teilungs- und Verschmelzungsmethoden liegt.
Wichtig ist der Hinweis, dass höhere Dimensionen sich nicht negativ auf die Bucketbelegung auswirken, da höhere Dimensionen auch mehr Verschmelzungsmöglichkeiten mit sich bringen.

Das wachsende File

Das Grid File werde mit Einträgen überhäuft (10.000 Datensätze). Die Simulation wurde für zwei verschiedene Bucketkapazitäten durchgeführt. Sowohl für 50 als auch für 100 Datensätze pendelte sich die durchschnittliche Bucketbelegung bei 70% ein (vgl. *Abbildung* 6). Dieser Wert erinnert an den „magischen Wert“ der oft in theoretischen Analysen auftaucht: *ln 2 = 0.6931...*
An dieser Stelle sei angemerkt, dass wenn man die Teilungsstrategie dahingehend verändert, dass ein überlaufendes Bucket nicht auf zwei sondern drei Buckets aufgeteilt wird, die Bucketbelegung auf 39% sinkt.
Da sich mehrere Grid Blocks ein Bucket teilen können, kann man nicht von dem Wachstum der Buckets auf das des Grid Directory schließen. Auch ist das Grid Directory deutlich für bestimmte Faktoren anfälliger als Buckets. So wächst das Directory bei korrelierten Attributen deutlich schneller an als die Anzahl der Buckets. Ähnlich verhält es sich, wenn

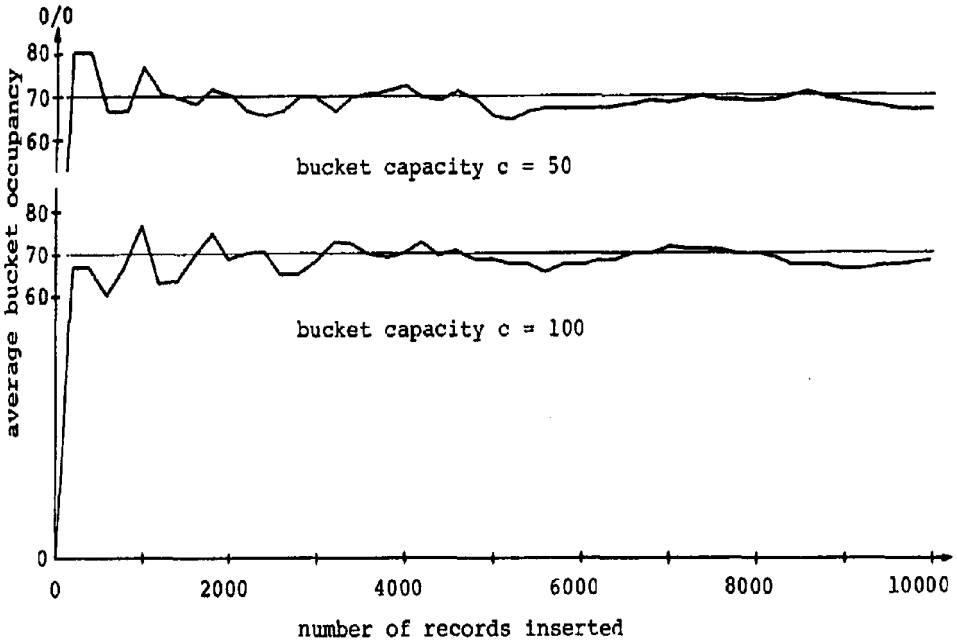

Abbildung 6: Die durchschnittliche Bucketbelegung eines wachsenden File. 10000 Datensätze werden bei einer Bucketkapazität von 50 bzw. 100 eingefügt.

bei einer kleiner Bucketkapazität der Großteil der Datensätze im selben Bereich liegen. Für die Visualisierung der Geometrie der Bucketregionen füllte man das Grid File mit 400 Datensätzen und begrenzte die Kapazität der Buckets auf 20. In *Abbildung* 7 erkennt man sehr gut wie es sich verhält, wenn die Datensätze einmal gleichverteilt und ungleichverteilt auftreten. Die 'Datenhaufen' werden elegant absorbiert.

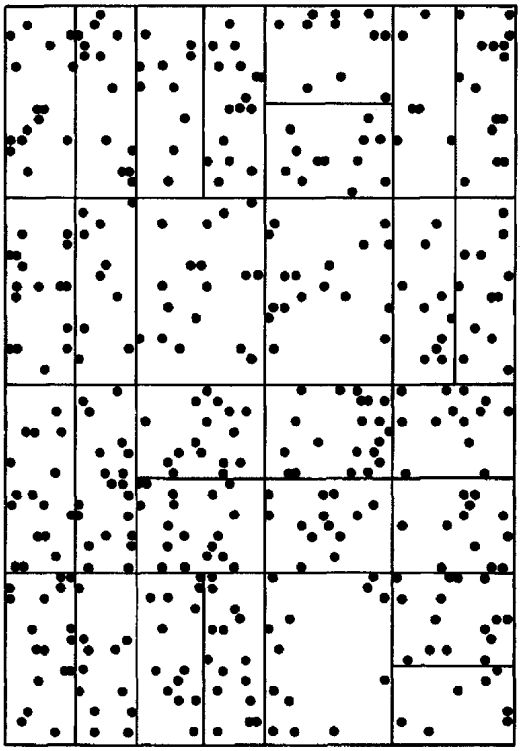

Abbildung 7a: Bucketregion mit gleichverteilten Daten.

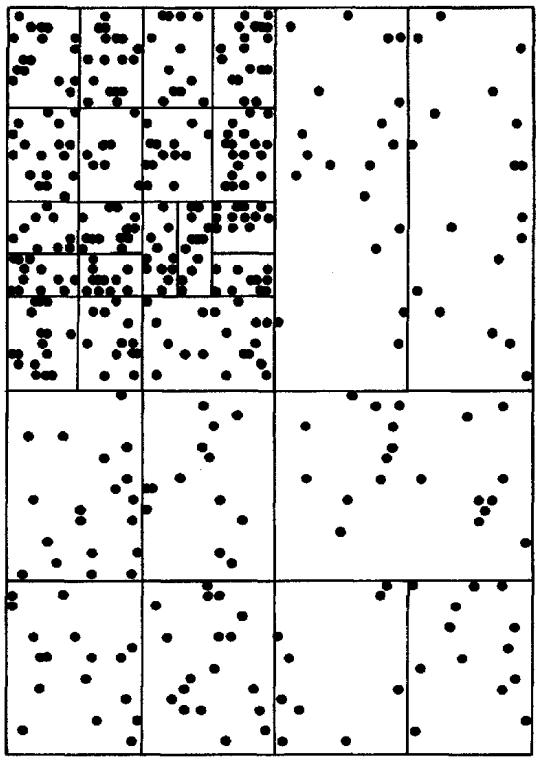

Abbildung 7b: Bucketregion mit ungleichverteilten Daten.

Das gleichgroß bleibende File

Beim gleichgroß bleibenden File ('steady-state file') werden viele Einfüge- und Löschoperationen so durchgeführt, dass die Anzahl der Daten im File konstant bleibt. Um die durchschnittliche Bucketbelegung zu analysieren betrachtet man nun die Verschmelzungsgrenzwerte, also die Belegung des resultierenden Buckets. Natürlich könnte man einen Verschmelzungsgrenzwert von 100% festlegen und würde damit auch eine hohe Bucketbelegung erhalten. Dennoch würde sich dadurch auch die Anzahl der Operationen deutlich erhöhen, weil vollere Buckets wiederum schneller überlaufen. Um die Anzahl der Operationen gering zu halten, empfiehlt sich ein Verschmelzungsgrenzwert.

Das schrumpfende File

Aus einem gefüllten Grid File werden kontinuierlich Daten entfernt. Dabei zeigt sich sehr bald, dass sich eine niedrige Bucketauslastung entwickelt. In diesem Fall wäre ein Verschmelzungsgrenzwert von 100% optimal, denn die durchschnittliche Bucketbelegung scheint nach vielen Löschoperationen immer noch zwischen 60 und 70% zu liegen.

5. Schlussfolgerung

Das Grid File eignet sich für große Datenmengen mit Daten, die weniger als 10 Attribute haben und deren Wertebereich groß und linear geordnet ist.

Das Grid File hat folgende Charakteristika:

- Symmetrische, anpassende Multikey-Datenstruktur
- Erfüllt die zwei Prinzipien, das 'Two-Disk-Access' Prinzip und das Prinzip der 'Efficient Range Queries with Respect to all Attributes'
- Hat ein Grid Directory zur Verwaltung der Korrespondenz
- Schneller Zugriff auf individuelle Datensätze
- Effiziente Bereichsanfragen
- Unempfindlich gegenüber 'Datenhaufen'
- Simulationen zeigen, dass die Speicherauslastung gut ist (70%).

6. Quellenangaben

- J. Nievergelt, H. Hinterberger (Institut für Informatik, ETH Zürich) K.C. Sevik (University of Toronto): An Adaptable, Symmetric Multikey File Structure, ACM Transactions on Database Systems, Vol. 9, No.1, March 1984, Pages 38-71.
- Die Abbildungen 6 und 7 (Seite 9) stammen aus der obigen Quelle.